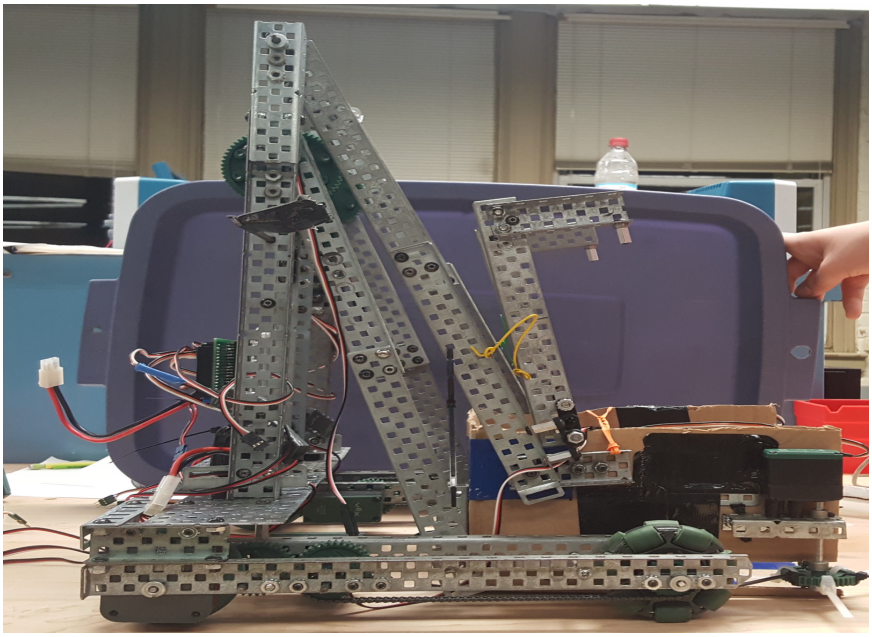
TEAM 10
RBE 1001 FINAL REPORT



Member	Signature	Contribution %
Caleb Wagner	*Caleb Wagner*	40
Dakota Payette	*Dakota Payette*	30
Xuhao Zhang	*Xuhao Zhang*	30

Grading	
Presentation	/20
Design Analysis	/30
Programming	/30
Accomplishment	/20
Total	/100

FEBRUARY 28, 2017

WPI

RBE 1001

Table of Contents

INTRODUCTION **1**

PRELIMINARY DISCUSSION **2**

AUTONOMOUS: 2

TELEOPERATION: 2

END OF PLAY: 2

PROBLEM STATEMENT **3**

PRELIMINARY DESIGNS **3**

DRIVETRAIN: 4

LIFT AND HANGING MECHANISM: 4

INTAKE: 5

ELECTRONICS AND SENSORS: 5

SELECTION OF FINAL DESIGN **6**

DRIVETRAIN: 6

LIFTING MECHANISM: 9

HANGING MECHANISM: 16

INTAKE: 16

ELECTRONICS AND SENSORS: 16

FINAL DESIGN ANALYSIS **17**

CENTER OF GRAVITY: 17

TIPPING FACTOR: 17

TRACTION FACTOR: 19

PUSHING FORCE: 20

LINEAR SPEED: 21

CHAIN LOAD: 21

PROGRAMMING ANALYSIS: 22

CUSTOM CIRCUIT ANALYSIS: 23

SUMMARY AND EVALUATION **25**

APPENDIX **27**

CODE: 27

Autonomous Code 1: Score in NEST (not implemented) *27*

Autonomous Code 2: Drive on Ramp *31*

Autonomous Code 3: Teleoperation *34*

Autonomous Code 4: Final RBE 1001 Template *36*

MOTOR DATA CHARTS: 41

VEX 3-Wire Motors: *41*

VEX 393 Motors: *41*

SCORING GUIDELINES: 42

PICTURES: 42

List of Figures

Figure 1.1: Field for project 1
Figure 4.1: Preliminary design of robot 3
Figure 4.2: Demonstration of hanging mechanism 5
Figure 4.3: Different intake considerations 5
Figure 5.1: Traction force on wheel 7
Figure 5.2: Output torque on wheel 8
Figure 5.3: Instant turning centers of four-bar 9
Figure 5.4: Power of four-bar 11
Figure 5.5: Four-bar FBD 12
Figure 5.6: Collector FBD 13
Figure 5.7: Arm FBD 14
Figure 6.1: Center of gravity location 17
Figure 6.2: Tipping factor FBD 18
Figure 6.3: Traction factor FBD 19
Figure 6.4: Pushing force FBD 20
Figure 6.5: Chain load FBD 22
Figure 6.6: LCD Schematic 24
Figure 5.7: Push Button Schematic 25
Figure A.1: VEX 3-Wire Motor Data 41
Figure A.2: VEX 393 Motor Data 41
Figure A.3: Final Robot Size 42
Figure A.4: Final Robot Above 43
Figure A.5: Final Robot Raised 43

List of Tables

Table 1: Motor Speed and Current 14
Table 2: Motor Torque and Current 15
Table 3: Scoring Guidelines for Project 42

Introduction

The purpose of this project is to design and build a radio- and autonomously-controlled robot that includes in its design sensor-based autonomous behavior, at least one non-trivial power transmission and lifting device, and at least one custom electronic circuit. Each component will demonstrate competence in the fundamental disciplines that comprise the robotics engineering field. The robot will use the field shown below (Figure 1.1) to demonstrate its proposed abilities. Points will be awarded for completing various tasks centered around picking up and scoring wooden eggs.

The robot must be no larger than 15.25” by 15.25” by 18” and weigh no more than 10lbs. It will be constructed from primarily VEX pieces and motors with an Arduino Mega as the microcontroller.

A twenty second autonomous period will start the challenge. During this time points can be awarded for depositing preloaded eggs in the COOP, PEN, or NEST, driving on the RAMP, or pushing the PEN. After the autonomous period will be a two minute teleoperation period, during which the robot will collect wooden eggs scattered about the field and score them in the COOP, PEN, or NEST. Additional points will be added at the end of the teleoperation portion if the robot is not touching the carpet. A summary of the methods of scoring can be found in the Appendix, as well as a description of the various components that comprise the field.

Figure 1.1:

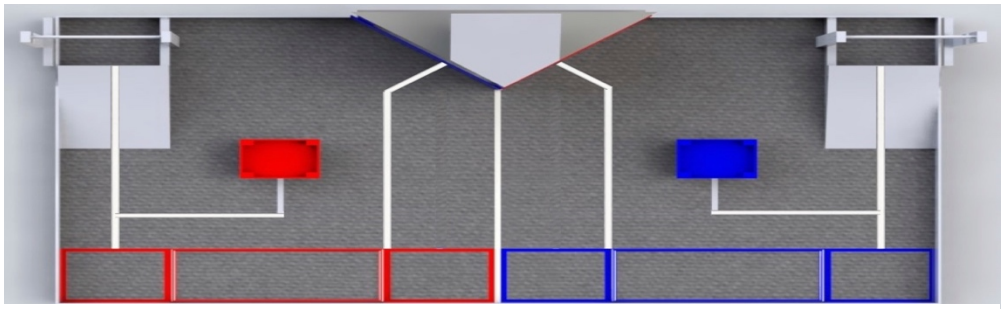


Figure 1.1: Field for project

Preliminary Discussion

The goal of this robot is to collect as many eggs as possible as fast as possible. Once an adequate number of eggs have been collected, the eggs will be scored in the NEST. For this reason, speed will be valued for the drivetrain over torque.

Autonomous:

The primary goal of the twenty second autonomous period is to score both of the pre-loaded eggs in the NEST to accumulate a total of six points. This will be accomplished through the use of line sensors which will follow the designated white lines on the field for a certain distance, turn, and then continue traveling straight until a limit switch at the front of the robot is triggered. The robot will align itself into scoring position and deposit the eggs in the NEST.

After the two eggs have been scored, the robot will turn towards the RAMP and try to drive on the RAMP for an additional five points. This will be accomplished through quadrature encoders and line followers. Nevertheless, due to the limited amount of time for the autonomous period, the primary goal will be to score the two pre-loaded eggs in the NEST.

Teleoperation:

The primary goal of the teleoperation portion of the competition will be used to collect as many eggs as possible and score the eggs in the NEST. While the collection of eggs will be performed through operator control, the actual scoring of the eggs will be performed autonomously with occasional operator interrupts for adjustments. The operator will drive the robot into the NEST, triggering the limit switch. The robot will then align itself and deposit the eggs it is carrying.

End of Play:

The primary goal of the end of play will be to hang on the PERCH. This process will be made as efficient as possible in order to allow for increased time for collecting eggs.

Problem Statement

The goal of this project is to build a robot that can collect eggs and subsequently score them in the most efficient manner possible. An additional goal is to provide an academic reason for the design and selection of all components of the robot as well as demonstrate an understanding of the mechanical, electrical, and programming subsets that go into designing a robot.

Preliminary Designs

Based upon the various criteria for competition as well as the list of resources allocated for this project, designs were based upon the design goals. As such, they include a drivetrain, lift mechanism, hanging mechanism, intake, and various electronics and sensors. Different designs for each were discussed and decided upon accordingly (Figure 4.1).

Figure 4.1

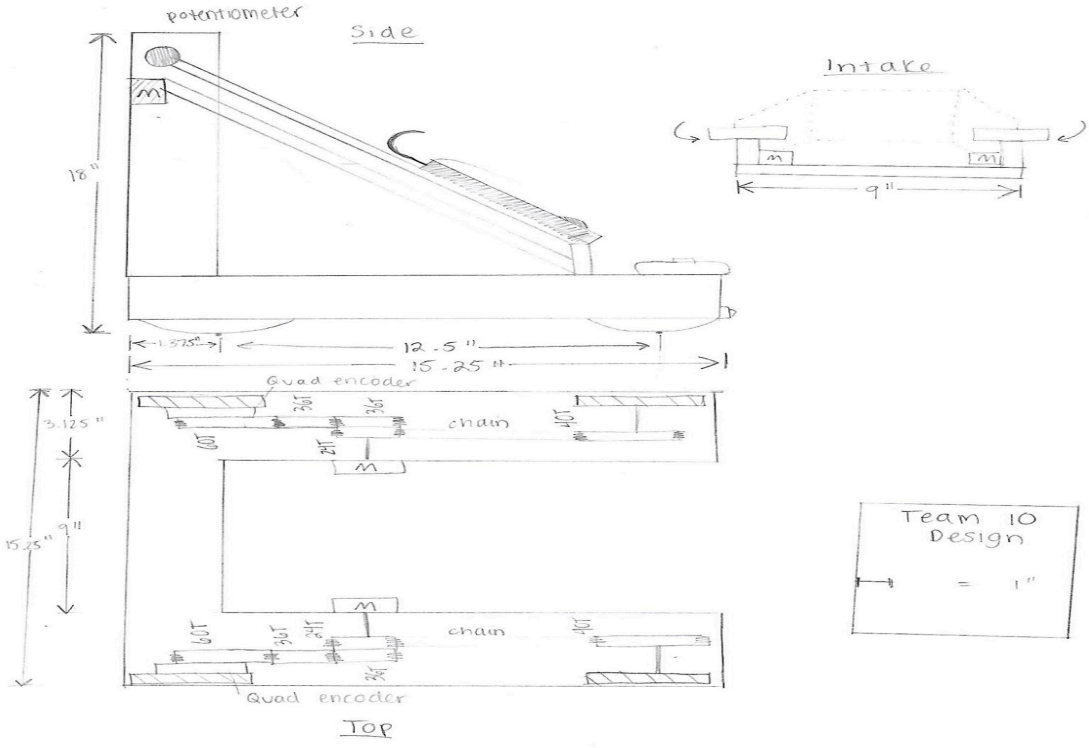


Figure 4.1: Preliminary design of robot

Drivetrain:

The drivetrain will consist of four 2.75” diameter wheels driven by two 393 VEX motors. The front two wheels will be Omni wheels in order to assist the robot in turning faster. The rear wheels will be connected to the motors through gears, while the front wheels will be connected to the motors through a chain and sprocket in order to accommodate the size of the drivetrain. Gearing will be decided based upon the combination that provides speed as well as an appropriate amount of torque for climbing the RAMP and for interactions with other robots and/or game pieces.

Other drivetrain ideas included the use of four-wheel drive with two more motors that are geared directly to the front wheels as opposed to using the chain and sprocket. This was decided against based upon the current draw of additional motors. Different wheel sizes were considered but due to parts made available for the project as well as restrictions on the size of the drivetrain, 2.75” wheels were decided upon.

Lift and Hanging Mechanism:

The lift mechanism for the robot will be a four-bar that has two main positions: the first on the ground in order to collect eggs, and the second in the air to deposit the eggs in the NEST. A transmission will be added in order to slow down the speed of operation. Other mechanisms discussed included an elevator with pulleys, a rack and pinion with linear slides to lift the robot, as well as a pneumatic lift. The four-bar was chosen for its simplicity and low weight.

The hanging mechanism will use the four-bar to hang on the PERCH. A hook will be attached to the follower of the four-bar to provide additional height. This attachment will be connected to the bottom of the crank with a string. When the four-bar is raised, the hook will rise

into position as well, giving the additional height required (Figure 4.2). A locking mechanism will lock the four-bar into place to remove the strain on the motor.

Figure 4.2

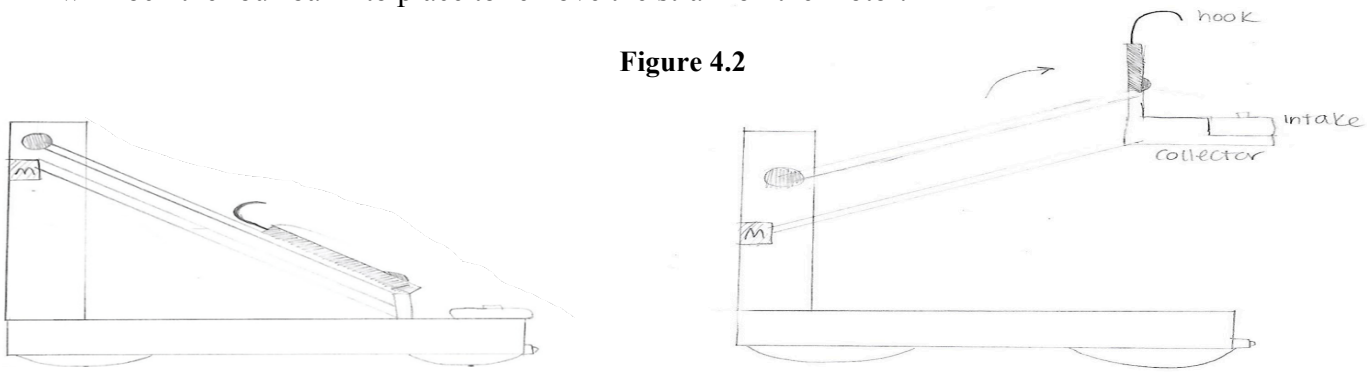


Figure 4.2: Demonstration of hanging mechanism

Intake:

Various intake designs were considered to increase the efficiency of egg collection. The considerations included spinning wheels, a spinning brush, and flaps controlled by servos that push the balls into a collector (Figure 4.3).

Figure 4.3

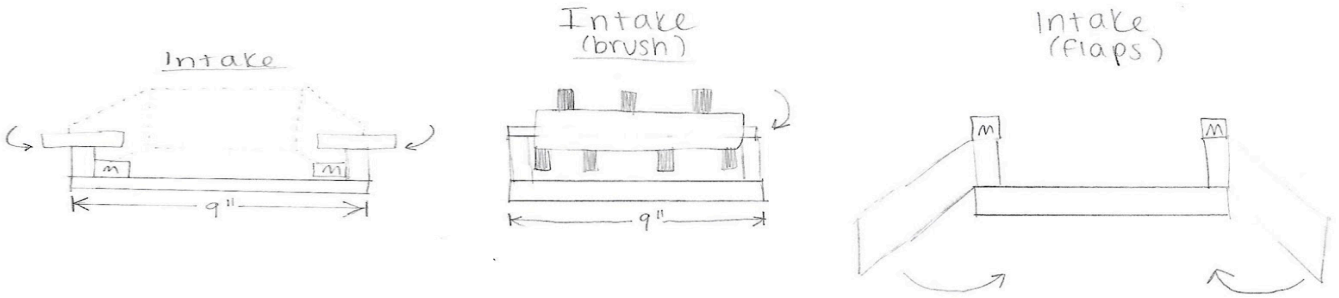


Figure 4.3: Different intake considerations

Electronics and Sensors:

Various sensors were considered for this project, including quadrature encoders for drift correction, a potentiometer to determine the position of the four-bar, and a limit switch to determine when something is in front of the robot. The custom circuit will be the LCD screen that will print to screen the functions that the robot is performing in order to assist in debugging.

A major consideration will be the max 7.5A of the Arduino Mega, as the several motors and sensors are expected to draw current somewhere around 7A.

Selection of Final Design

Drivetrain:

As desired, the final dimensions of the drivetrain for the robot were 15.25" by 15.25". The front wheels were connected to the motors using a chain and sprocket, while the back wheels were geared to the motors. Only two motors were used in order to limit current draw.

Originally, the drivetrain was built using C channel in order to provide increased stability. Due to the weight constriction of 10lbs, however, the C channel was substituted for the much lighter L channel. This saved about 1lb of weight. The large size of the drivetrain was effective at accommodating the gears, chain and sprockets, and optical quadrature encoders, in addition to affording enough space in the middle for the collector and intake.

After determining the layout for the drivetrain that would make the most efficient use of space, the next biggest consideration for the drivetrain was the selection of motors. In order to do this, the tractive force of the robot was calculated in order to determine the forces on each wheel. Because the robot uses all-wheel drive, the wheels could be modeled as a single wheel with respect to friction. The coefficient of friction was determined to be approximately 0.7 based off of research and previous experimentation of VEX wheels on carpet, while the weight of the robot was assumed to be 10lbs. Accordingly, a free-body diagram of the wheel was drawn and forces were summed in the x- and y-directions as follows (Figure 5.1):

Figure 5.1

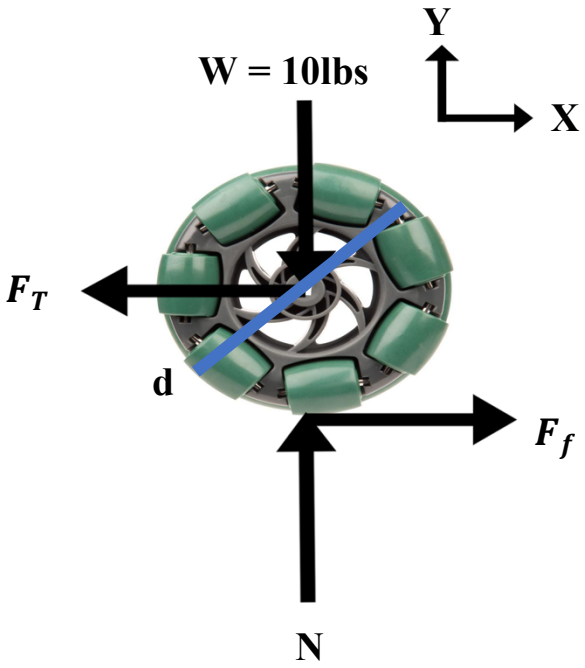


Figure 5.1: Tractive force on wheel

$$\sum F_x = 0 = F_f - F_T \qquad \sum F_y = 0 = N - 10\ lbs$$
$$F_f = F_T \qquad N = 10\ lbs$$

Solving for the tractive force gave:

$$F_T = \mu N = (0.7)(10lbs) = 7lbs$$

With the calculated tractive force of 7lbs, it was possible to determine the power that the drivetrain motors would need to move the robot at a minimum speed of 0.5 feet per second. This speed was determined after desiring a relatively fast speed for the robot while not wanting to sacrifice stability or afford the potential to drop eggs while driving. Power was calculated as follows:

$$P = \frac{Fd}{t} = \frac{(7lbs)(0.5ft)}{1s} * \frac{746\ watts}{550ft\ lbs} = 4.8\ watts$$

This showed that each of the two motors used in the drivetrain must be able to produce 2.4 watts to drive at this speed. This immediately excluded the use of the 3-wire motors as they would not be able to produce the appropriate power (see Appendix for 3-wire motor data). The 393 motors were chosen instead as operating at 2.4 watts for each motor meant that the motor would operate at around peak efficiency (see Appendix for 393 motor data).

The next step after determining which motors to use was determining the required wheel size and gear ratio in order to provide the appropriate speed and torque. Because the input torque for both wheels would be the same, the only change in using different size wheels would be the amount of traction force that each wheel would provide. Accordingly, a free-body diagram was drawn for VEX 2.75" wheels and VEX 4" wheels (Figure 5.2):

Figure 5.2

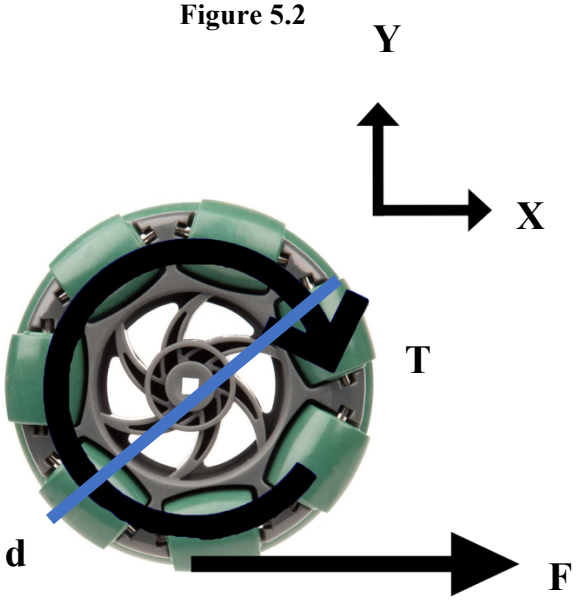


Figure 5.2: Force of wheel

$$\sum M_z = 0 = T - (\frac{d}{2})F$$

$$F = (\frac{d}{2})T$$

$$F_{4\ inch\ wheel} = \left(\frac{4in}{2}\right)T$$

$$F_{2.75\ inch\ wheel} = \left(\frac{2.75in}{2}\right)T$$

Because each wheel would have the same torque, having a larger wheel means less tractive force. As a result, the 2.75” wheels were chosen over the 4” wheels in order to provide better traction.

Using a 1:1 gear ratio, the 2.75” wheels would produce the necessary torque to overcome the friction force while drawing about 3A from the battery (see Appendix for 393 motor data). The 3A was considered an acceptable tradeoff as the 1:1 gear ratio gave the desired speed for the robot (see Final Design Analysis: Linear Velocity to see calculation of linear velocity). Furthermore, the use of 36 tooth gears to gear the wheels made the most efficient use of space in the drive train.

Lifting Mechanism:

The four-bar lifting mechanism was selected for the final design. Three different positions were determined in order to analyze the instantaneous turning center and the displacement vector to ensure that the collector would angle down as the four-bar was raised, thereby aiding in the scoring of eggs by facilitating the removal of eggs from the collector (Figure 5.3). Note the diagram does not show position 3 as the displacement vector of the collector is unimportant for hanging purposes.

Figure 5.3

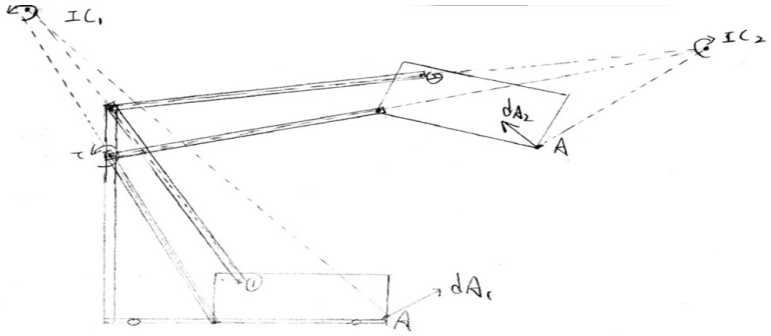


Figure 5.3: Instant Turning Centers of Four-Bar

At the third position when the four-bar is almost vertical, the lower link lines up with the collector such that they form a toggle position. It was determined that the lower link should be powered in order to prevent the upper link from needing to reverse direction in order to ensure that the collector rotates in the desired manner and not have any negative consequences on the four-bar.

Due to the nature of parts made available for the project, the height of the four-bar was initially lowered from 18” to 16”. No VEX parts were found to be long enough for the four-bar to be 15” long to accommodate a 7” by 9.5” collector and to raise to the proper height to score in the NEST and hang on the PERCH. However, lowering the four-bar meant that 15” four-bar arm rested at a much larger angle than at 18”, such that the collector had to be pushed further out of the drivetrain in order to accommodate the four-bar arm. The sacrifice in space for the collector and intake was found to be unacceptable. As a result, the four-bar was moved back up to 18”, and two smaller L channels were attached together using standoffs to provide the desired 15”. While this method provides less structural support for the arm than a singular L channel, the increased space made this a reasonable tradeoff. The top arm was connected slightly further along the collector as opposed to the bottom arm in order to rotate the box downwards as the four-bar was raised.

The next step in designing the four-bar was determining which motors to use to lift the arm. The desired speed of the arm was about 5RPM. The weight of the four-bar was found to be 1.3lbs and the weight of the intake with a few preloaded eggs was found to be about 1.2lbs. Accordingly, the power required for the four-bar was calculated by converting the angular speed of the four-bar to linear speed and multiplying by the weight of the arm and intake (Figure 5.4):

Figure 5.4

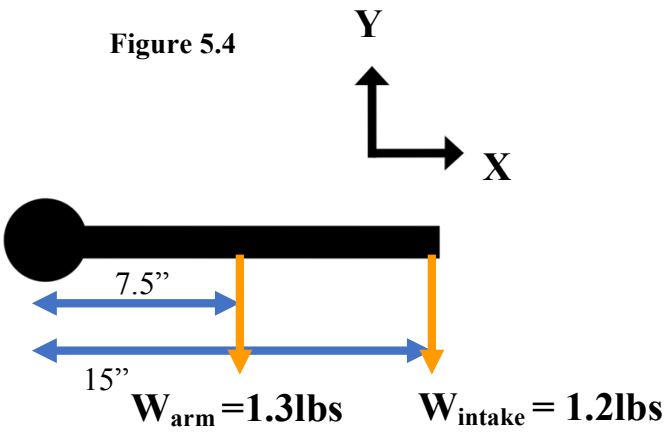


Figure 5.4: Power of four-bar

$$\square = 5RPM = 0.5\,{}^{rads}/_{s}$$

$$V_{arm} = \omega r_{arm} = \left(0.5\,{}^{rads}/_{s}\right)(7.5in) = 3.75in/s$$

$$V_{intake} = \omega r_{intake} = \left(0.5\,{}^{rads}/_{s}\right)(15in) = 7.5in/s$$

$$P = (V_{arm})(W_{arm}) + (V_{intake})(W_{intake}) = (3.75\,in/s)(1.3lbs) + (7.5\,in/s)(1.2lbs)$$

$$P = \frac{14in\ lbs}{s} = \left(1.2\frac{ft\ lb}{s}\right) * \frac{746\ watts}{550ft\ lbs} = 1.6\ watts$$

Next the power required to hang was calculated assuming the weight of the robot to be 10lbs and selecting the speed at which the robot rises off the ground to be 0.5ft per 3 seconds:

$$P = \frac{Fd}{t} = \frac{(10lbs)(0.5ft)}{3s} * \frac{746\ watts}{550ft\ lbs} = 2.26\ watts$$

As such, it was determined that two 3-wire motors would be able to provide the necessary power to lift the four-bar and hang (see Appendix for 3-wire motor data). However, after experimenting with the 3-wire motors, it was determined that the given motors were not operating at the same values as the motor data chart and did not in fact have the required power to work several times. As a result, it was decided that one 393 motor would be used in order to ensure repeated success and reliability, as well as provide the required power and torque.

Finally, the gear ratio was calculated in order to move the four-bar at the desired speed in addition to providing enough torque to lift the intake. A free-body diagram was first drawn of the entire four-bar (Figure 5.5):

Figure 5.5

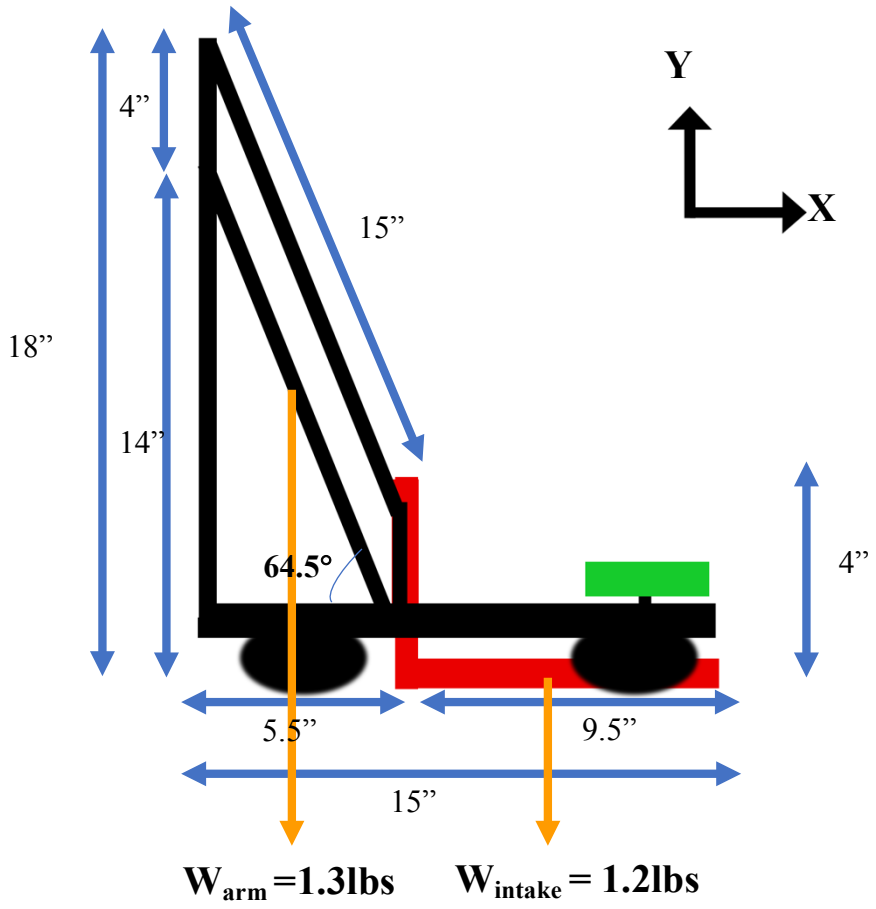


Figure 5.5: Four-bar free-body diagram

From this diagram, the angle of the four-bar was determined to be 64.5°. In order to determine the forces acting upon the intake, a free-body diagram was drawn of the intake and then the equations of equilibrium were used to find the values of F_x and F_y for the intake (Figure 5.6):

Figure 5.6

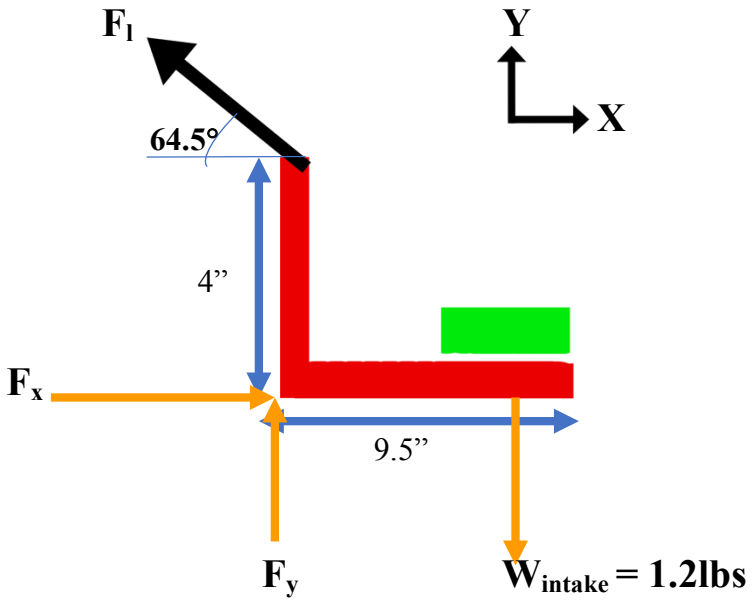


Figure 5.6: Collector free-body diagram

$$\sum M_z = 0 = F_l \cos(64.5)\,(4) - (1.2lbs)(8in)$$

$$F_l = 5.6lbs$$

$$\sum F_x = 0 = F_x - F_l \cos\,(64.5)$$

$$F_x = 2.4lbs$$

$$\sum F_y = 0 = -F_y + F_l \sin(64.5) - (1.2lbs)$$

$$F_y = 3.8lbs$$

Finally, a free-body diagram was drawn of the arm of the four-bar and the summation of torques was calculated in order to determine the output torque of the four-bar (Figure 5.7):

Figure 5.7

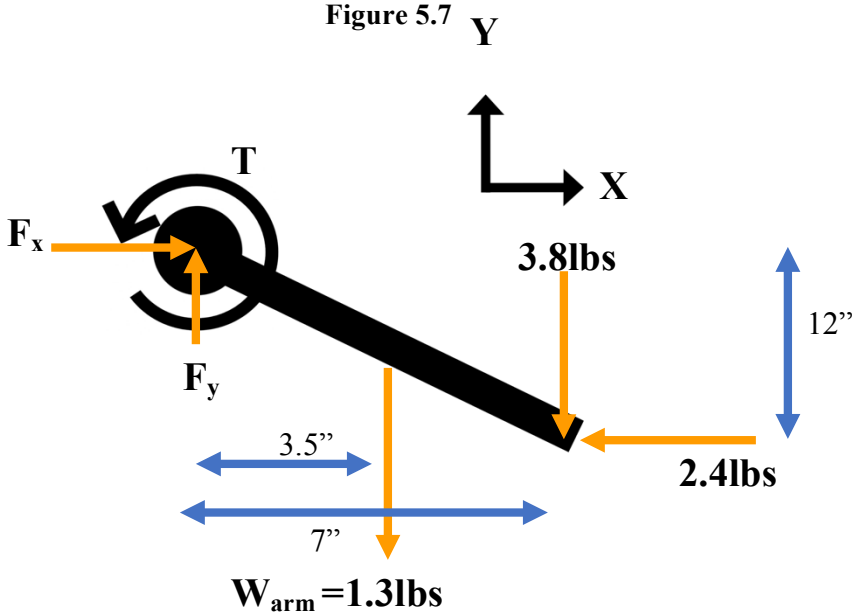


Figure 5.7: Free-body diagram of arm

$$\sum M_z = 0 = T_{out} - (2.4lbs)(7\text{in}) - (3.8lbs)(12in) - (1.3lbs)(3.5)$$
$$T_{out} = 66.95\ in\ lbs$$

Using this information, the gear ratios were calculated. It was decided that the motor should not exceed 2A so that the motor does not operate outside its peak efficiency (see Appendix for 393 motor data), in addition to not wanting to draw too much current so as to exceed the 7.5A limit of the Arduino Mega when all of the motors in the robot are coupled together. The speed requirement for the lift was determined to be about 5RPM as this speed gave enough control over raising the four-bar. Linear interpolation was used to determine the speed of the motor at 2A:

Table 1	
Speed (RPM)	Current (Amps)
60	2.142
X	2
67	1.847

Table 1: 393 Motor speed and current

$$\frac{x-67}{(60-67)} = \frac{(2-1.847)}{(2.142-1.847)}$$

$$x = 63.4RPM$$

The gear ratio was then calculated using the speed of the motor:

$$\frac{N_{out}}{N_{in}} = e = \frac{5}{63.4} = 0.8$$

Next, the gear ratio was calculated using torque. Linear interpolation was used to determine the torque of the motor at 2A:

Table 2	
Torque (in lbs)	Current (Amps)
5.9	2.142
X	2
4.92	1.847

Table 5.2: 393 Motor torque and current

$$\frac{x-4.92}{(5.9-4.92)} = \frac{(2-1.847)}{(2.142-1.847)}$$

$$x = 5.4in\ lbs$$

The gear ratio was calculated using the torque of the motor. The efficiency of the gears was assumed to be 70% based off of research and experimentation with VEX gears:

$$\frac{T_{in}}{T_{out}}\eta = e = \frac{5.4}{66.95}0.7 = 0.6$$

As such, the lower gear ratio of 0.6 was the ratio that was used to determine what size gears to use. Accordingly, a two-stage transmission was used, consisting of a 12:36 gear ratio for the first stage and a 12:60 gear ratio for the second stage.

Hanging Mechanism:

The hanging mechanism of the four-bar was changed from using a singular hook to two hooks. The reason for the change was because the collector was made from cardboard in order to conserve weight, and as such, a hook could not be attached to the cardboard in such a way that the cardboard could support 10lbs. Therefore, two hanging mechanisms were put on each of the four-bar linkages. The dual hook system provides greater stability when hanging than the use of only one hook. Attaching wire to the bottom of the hook and the bottom of the four-bar allowed to the hooks to rise into position as the four-bar was raised, such that they provided the extra height required to hang on the PERCH.

Intake:

After experimenting to determine the most effective design for the intake, it was decided that two spinning wheels were the most effective. Due to the size constrictions of the robot, the wheels were to gears with holes drilled in them to have a smaller rotating mechanism for the intake. Attached to the gears were zip-ties that were chosen for their rigidity as well as their ability to bend once a certain threshold was met. This ensured that the eggs would be pushed into the collector, while at the same time protecting the intake if it were to collide with a wall or obstacle. The zip-ties also allowed for a further consolidation of space.

The collector was also originally built as metal. However, due to the weight constriction of 10lbs, the collector was replaced with a cardboard box. This saved about 1.2lbs in weight.

Electronics and Sensors:

The LCD was used as the custom circuit. One change to the schematic of the circuit was the addition of a pushbutton in order to initialize certain protocols of the LCD. The pushbutton

was chosen over the use of the joystick controller for initialization in order to keep the joystick for controlling the robot rather than the LCD.

Final Design Analysis

Center of Gravity:

The center of gravity of the robot was found through the use of a scale that calculated the location of the center of gravity with respect to the origin. As such, the center of gravity was found to be at 7.5” on the x-axis, 4.5” on the y-axis, and 6.5” on the z-axis. A diagram showing the location of the center of gravity can be found below (Figure 6.1).

Figure 6.1

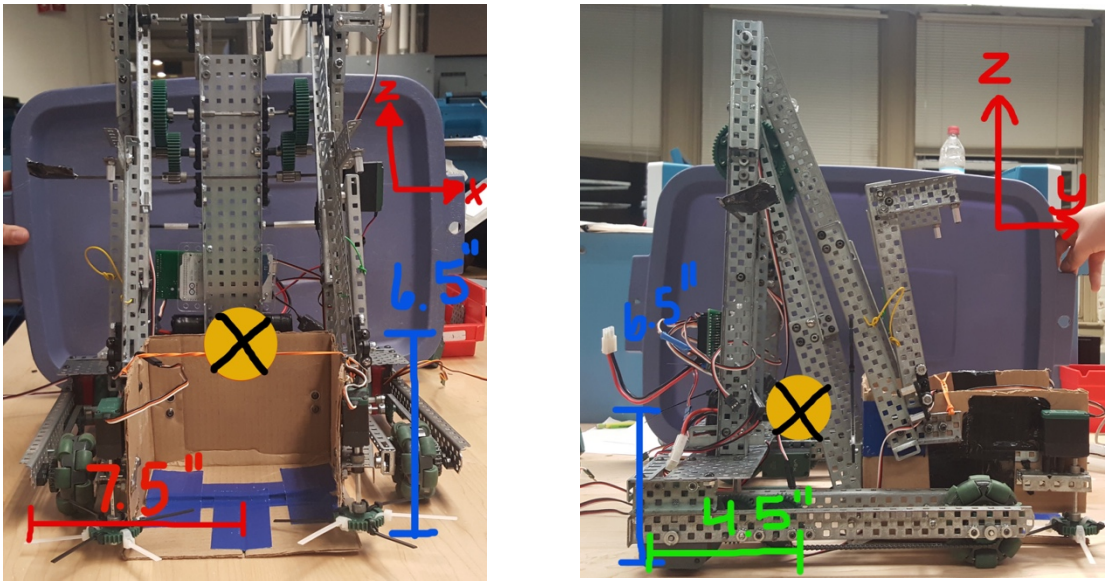


Figure 6.1: Center of gravity location

Tipping Factor:

The tipping factor of the robot was found by drawing a free-body diagram of the robot on an incline and using the equations of equilibrium (Figure 6.2):

Figure 6.2

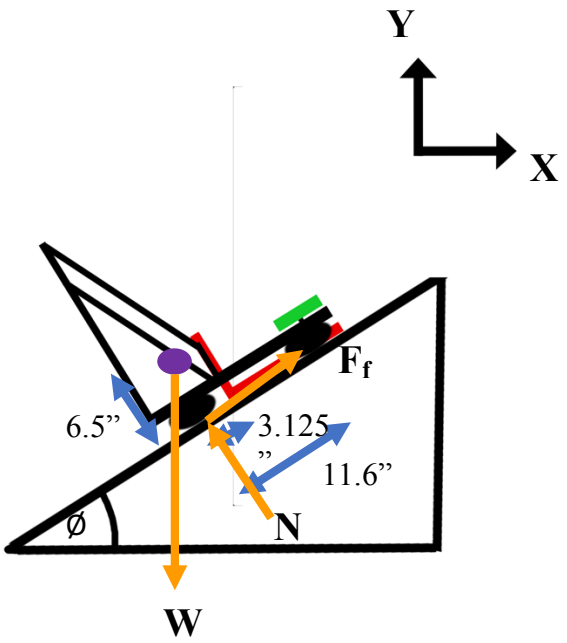


Figure 6.2: Tipping factor free-body diagram

$$\sum F_y = 0 = N - W\cos(\theta) \qquad \sum F_x = 0 = W\sin(\theta) - F_f$$
$$N = W\cos(\theta) \qquad F_f = W\sin(\theta)$$

$$\sum M_z = 0 = (W\sin(\theta))(6.5in) - (W\cos(\theta))(3.125in)$$

$$(W\sin(\theta))(6.5in) = (W\cos(\theta))(3.125in)$$

$$\tan(\theta) = \frac{3.125in}{6.5in} = 26°$$

26° is a reasonable number because it is greater than the 11° needed to climb the RAMP. Furthermore, raising the four-bar will only move the center of gravity further along the x-axis, thereby increasing the angle at which the robot tips over. Since the four-bar can only go from approximately 25° to 155°, the center of gravity will never shift backwards. Therefore, the robot is actually able to drive up ramps steeper than 26° depending on the angle of the four-bar.

Traction Factor:

The traction factor of the robot was found by drawing a free-body diagram of the robot on an incline and finding the equations of equilibrium (Figure 6.3). It was assumed that the Omni wheels in the front had the same coefficient of friction when moving forwards as the back wheels, as only when turning does the coefficient of friction acting on the Omni wheels lessen.

Figure 6.3

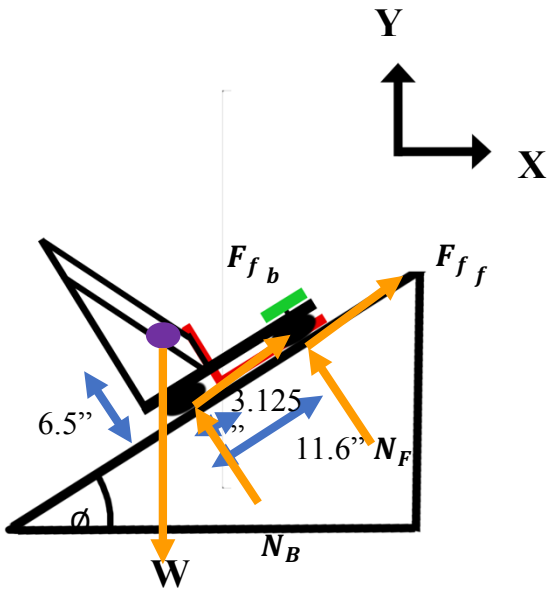


Figure 6.3: Traction factor free-body diagram

$$\sum F_y = 0 = -W\cos(\theta) + N_F + N_B \qquad \sum F_x = 0 = -Wsin(\theta) + F_{f\ f} + F_{f\ b}$$

$$N_F + N_B = W\cos(\theta) \qquad F_{f\ f} + F_{f\ b} = \mathrm{W}\sin(\theta)$$

$$F_{f\ f} = \mu N_F = (1)N_F$$

$$F_{f\ b} = \mu N_b = (1)N_b$$

$$N_F + N_B = F_{f\ f} + F_{f\ b}$$

$$W\cos(\theta) = \mathrm{W}\sin(\theta)$$

$$\theta = 45°$$

The robot therefore loses traction at 45°. As a result, the robot will tip over before it loses traction.

Pushing Force:

To calculate the pushing force of the robot, the stall torque of the motors was divided by the radius of the wheels. Because the Arduino Mega cannot exceed 7.5A, the stall torque was calculated for each motor at 3.75A, giving a stall torque of approximately 11in lbs per motor (Figure 6.4). Using the radius of the wheels as 2.75", it was possible to calculate the maximum pushing force of the robot as follows:

Figure 6.4

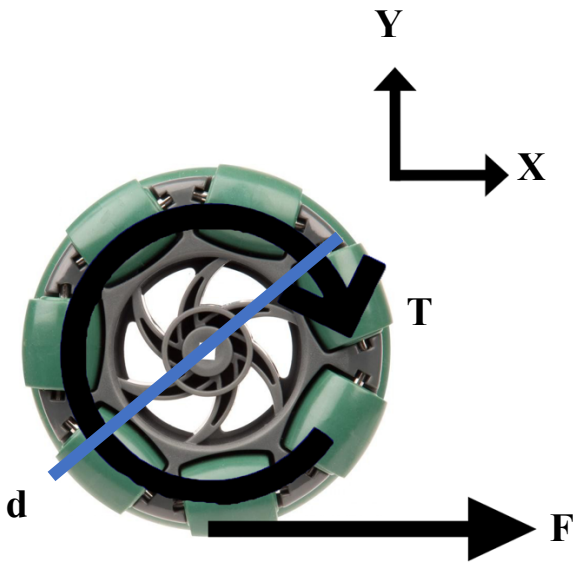


Figure 6.4: Pushing force free-body diagram

$$\sum M_z = 0 = T - (\frac{d}{2})F$$

$$T = (\frac{d}{2})F$$

$$\frac{2(22\ in\ lbs)}{2.75in} = 16lbs$$

The robot can therefore push a maximum of 16lbs. However, it is highly unlikely that the robot will ever need to push this much weight, and as a result, the robot should be able to push all items on the field including the PEN and other robots if necessary without exceeding the maximum amperage of the Arduino.

Linear Speed:

The linear speed of the robot was calculated by converting the rotational speed of the wheels to linear speed. As determined when calculating the power of the drivetrain, each motor will operate at 1.5A (3A total). As such, the angular velocity of the motors was found to be roughly 73RPM or 7.6 rads/s. Converting this to linear velocity and using the radius of the wheel as 1.375" gives:

$$v = \omega r = (7.6\ rads/s)(1.375in) = 10.45\,in/s = 0.9\,ft/s$$

As a result, the robot is able to travel 0.9 ft/s, which is greater than the minimum desired speed of 0.5 ft/s.

Chain Load:

In order to ensure the that the chain would not break in the drivetrain, a free-body diagram of the chain was drawn and the equations of equilibrium were used to calculate the minimum size of the sprocket (Figure 6.5). The output torque of the motors was found to be 9.6 in lbs. The breaking strength of the VEX chain was found to be 50lbs. Using these numbers gave:

Figure 6.5

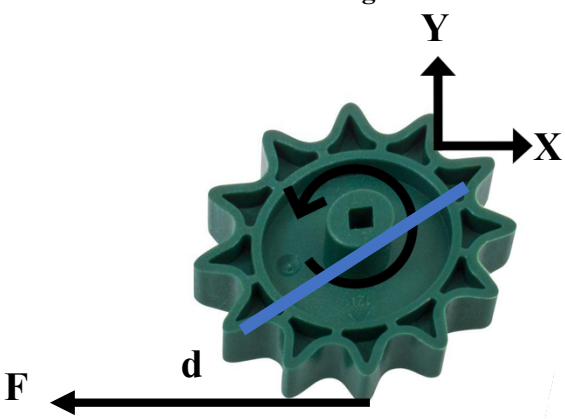


Figure 6.5: Chain load free-body diagram

$$\sum M_z = 0 = T - (\frac{d}{2})F$$

$$T = (\frac{d}{2})F$$

$$9.6in\ lbs = \left(\frac{d}{2}\right)(50lbs)$$

$$d = 0.384in$$

As a result, using the 24T sprocket which has a diameter of 1.18" gives the chain a safety factor of 3.

Programming Analysis:

The autonomous section of the program makes use of the optical quadrature encoders and line followers in order to drive straight and follow the white lines on the field. Both are controlled using PID in order to provide the greatest level of accuracy when driving and to ensure that the robot has built in course correction. The autonomous program begins with the robot driving from the starting zone for two feet and then turning to either the left or the right, driving forward, and then turning to the left or right again. The direction of the turn depends on which starting zone the robot is starting from. Two different versions of the program have the robot turning either to the left or right, driving straight for a foot, and then turning to face the

NEST. The robot drives straight using the optical quadrature encoders until the bump sensor on the front of the robot is trigged by hitting the NEST.

Once in front of the NEST, the robot backs up and positions itself directly in front of the scoring area of the NEST. Using the potentiometer, the four-bar rises to a predetermined height and releases the two preloaded eggs by reversing the direction of the intake such that the intake spins outwards. The robot then lowers the four-bar and waits until the end of the autonomous section. By staying right underneath where the RAD is located, the robot should be able to catch some of the eggs that fall off the RAD at the start of the teleoperation portion.

For debugging purposes, the LCD screen was used to display the code that the program was running through, in order to determine mistakes or flaws in the code.

The teleoperation code uses the joystick to control the robot. The left and right joystick control the drivetrain while the left and right triggers control the position of the four-bar (used for manual overrides). Once the teleoperation program begins, the intake beings spinning to facilitate the collection of eggs. This allows the robot to always be able to collect eggs as well as keeps the eggs in the collector while moving around. Pressing the second left trigger reverses the direction of the intake so as to score the eggs. Pressing button 1 on the joystick raises the four-bar to the height of the NEST so as to easily score on the NEST without having to manually raise the height. Button 2 raises the four-bar to the height of the PEN to score on the PEN. Both of these heights are controlled by the potentiometer. Pressing button 3 raises the four-bar to its maximum height in order to hang on the PERCH.

Custom Circuit Analysis:

The schematic of the custom circuit is as follows:

Figure 6.6

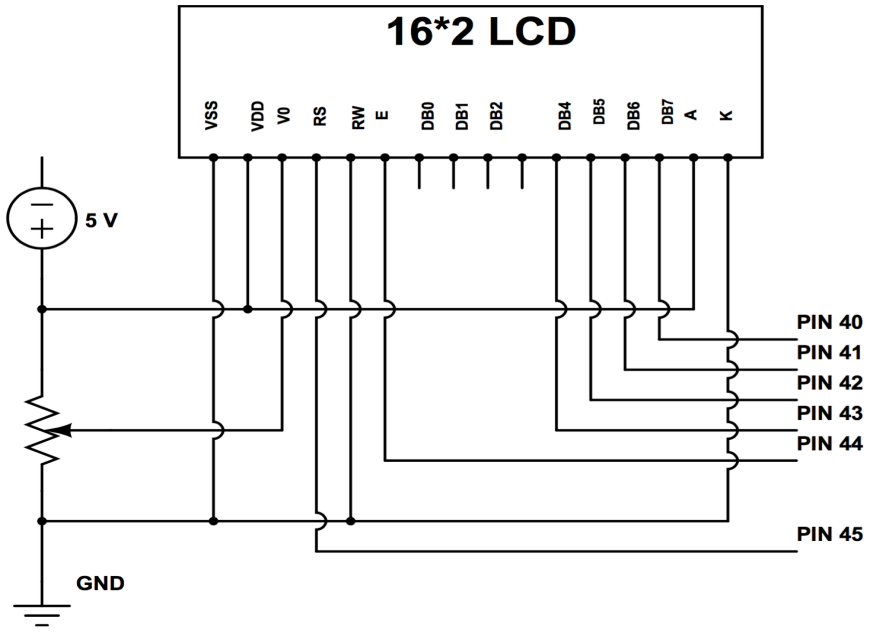


Figure 6.6: LCD Schematic

The Liquid Crystal Display (LCD) is connected to the Arduino using Digital Ports 40 through 45. Attached to the LCD is a variable resistor that is used to adjust the contrast of the LCD so that the screen can be seen in different lights (Figure 6.6).

A push button is also connected to Digital Port 28 and ground. When the button is pressed, current flows through the push button as the circuit is now complete. However, whenever the button is not pressed, no current flows through the push button. The Arduino is able to measure these changes in order to determine whether or not the button is pressed, such that the push button can be used as a digital input for the Arduino as it has one of two states, on or off (Figure 6.7).

Figure 6.7

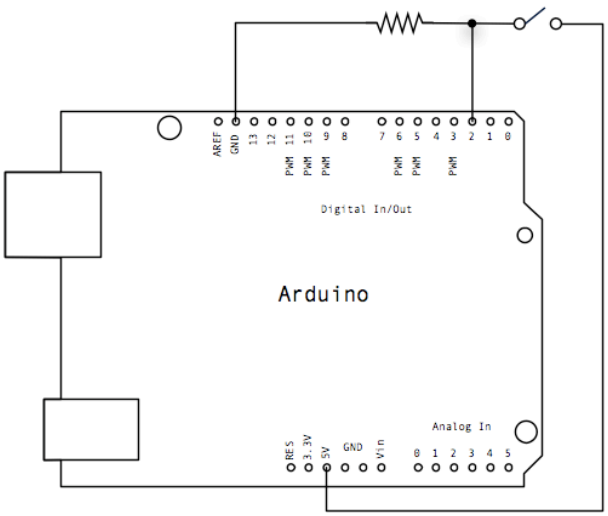


Figure 6.7: Push button schematic

Summary and Evaluation

Overall, the robot performed reasonably well. Although the robot was not able to accomplish all of the goals it set out to, it did complete most of them. See Appendix for pictures of final robot.

Mechanically, the robot operated incredibly smoothly. The overall mechanical aspect of the robot was well-designed and well-implemented. The four-bar consistently worked and was able to perform exactly as predicted by the math. One change that could have been made was to replace the cardboard box with something a little more durable. While the cardboard did last for all events, it also bent a little, especially when directly in front of the NEST. The cardboard was used to conserve weight for the robot, so it may have been better to have found alternative materials for parts, such as 3D printing some of them, in order to have a more durable container. The robot was not able to hang during competition, due to a desire to focus more on improving performance in the autonomous section and collecting eggs for the teleoperation section. The

hanging mechanism did extend and attach to the PERCH as expected, as well as did support the robot when placed directly on the PERCH. Accordingly, the hanging mechanism is still being viewed as a success as with a little more time, it would have been able to work for competition.

Programming and electronics was where the robot had a little bit more of a challenge. The line sensors used for the project were too sensitive to detect the difference in line values. As a result, the robot relied solely on the quadrature encoders in order to measure distance as well as to keep the robot driving straight. The encoders did not work exactly as expected and failed to keep the driving perfectly straight. This is being viewed as an electrical problem as switching the encoder to interrupt ports rather than digital ports may have improved their performance. As a result, the robot was not able to drive to the NEST and score in the NEST for competition. While it did work in practice, because the robot could not drive perfectly straight consistently, the program was not used for competition. Instead, the robot drove onto the ramp, where driving perfectly straight was less of a necessity than trying to align with the NEST.

As a result, this project is being considered a success. The robot was completed most of its goals, and most of the errors in this project can be attributed to programming rather than the overall design of the robot.

Appendix

Code:

Autonomous Code 1: Score in NEST (not implemented)

```
#include<Servo.h>
#include <LiquidCrystal.h>
#include <Encoder.h>


LiquidCrystal lcd(40, 41, 42, 43, 44, 45);

static enum stateChoices {
  DRIVING_FORWARD, BACKUP, TURNING, DRIVENEST, SCORING, STOPPED
} state;


int limitSwitchPort = 22;
int limitSwitchPort2 = 23;
int limitSwitch2;
int limitSwitch;
int Kp = 1;
int potPort = A10;
int potTarget = 390;
int DK = 3;

Servo rightmotor; // Servo object
Servo leftmotor; // Servo object
Servo armMotorLeft;
Servo armMotorRight;
Servo intakeMotorLeft;
Servo intakeMotorRight;

Encoder encoder1(28, 29);
Encoder encoder2(24, 25);

void setup() {
  Serial.begin(9600);
  leftmotor.attach(5, 1000, 2000);
  rightmotor.attach(4, 1000, 2000);
  armMotorLeft.attach(7, 1000, 2000);
  armMotorRight.attach(6, 1000, 2000);
  intakeMotorLeft.attach(9, 1000, 2000);
  intakeMotorRight.attach(8, 1000, 2000);
  state = DRIVING_FORWARD;
  pinMode(limitSwitchPort, INPUT);
  pinMode(limitSwitchPort2, INPUT);
  digitalWrite(limitSwitchPort2, HIGH);
  digitalWrite(limitSwitchPort, HIGH);
  lcd.begin(16, 2);
```

```
  lcd.print("Team 10 Rocks!");
  delay(1000);
  lcd.clear();
}

void loop() {
  DoDriving();                                          //drive and align with NEST
  ArmRaise();                                           //raise arm to max height
}

void DoDriving() {
  switch (state) {
    case DRIVING_FORWARD:
      setIntake(0, 180);
      lcd.setCursor(0, 0);
      lcd.clear();
      lcd.print("DRIVING_FORWARD");
      delay(100);
      limitSwitch = digitalRead(limitSwitchPort);       //read limit switch
      DriveStraight();                                          //drive straight
function
      if (limitSwitch == 0) {                           //if limit switch hit, at
NEST, change states
        state = BACKUP;
      }
      break;
    case BACKUP:
      lcd.setCursor(0, 0);
      lcd.clear();
      lcd.print("BACKUP");
      delay(100);
      Drive(40, 140);                                   //backup from NEST for
certain amount of time
      delay(700);
      state = TURNING;
      break;
    case TURNING:
      lcd.setCursor(0, 0);
      lcd.clear();
      lcd.print("TURNING");
      delay(100);
      Drive(90, 0);                                     //align robot with NEST so
intake can release eggs into NEST
      delay(500);
      state = DRIVENEST;
```

```
      break;
    case DRIVENEST:
      lcd.setCursor(0, 0);
      lcd.clear();
      lcd.print("DRIVENEST");
      delay(100);
      limitSwitch = digitalRead(limitSwitchPort);
      limitSwitch2 = digitalRead(limitSwitchPort2);
      Drive(150, 40);                                  //drive forward until left
or right limit switch is hit by NEST so robot is directly in front of NEST
      if (limitSwitch == 0 || limitSwitch2 == 0) {
        state = SCORING;
      }
      break;
    case SCORING:
      lcd.setCursor(0, 0);
      lcd.clear();
      lcd.print("SCORING");
      delay(100);
      Drive(90, 90);                                   //stop driving
      setIntake(180, 0);                               //reverse direction of
intake to score eggs
      delay(3000);
      state = STOPPED;
      break;
    case STOPPED:
      lcd.setCursor(0, 0);
      lcd.clear();
      lcd.print("STOPPED");
      delay(100);
      setIntake(90, 90);                               //stop intake
      break;
  }
}

void Drive (int leftValue, int rightValue) {
  leftmotor.write(leftValue);
  rightmotor.write(rightValue);
}


void DriveStraight() {
  int leftEncoderValue = encoder1.read();          //get values of both encoders
  int rightEncoderValue = encoder2.read();
  int difference = abs(leftEncoderValue) + abs(rightEncoderValue);
```

```
//calculate difference between the two values of the encoders
  if (difference > 20) {                          //wrap protection for encoders
    difference = 20;
  }
  if (difference < -20) {
    difference = -20;
  }
  Serial.println(difference);
  int drift = DK * difference;                    //use proportional control on
difference between encoder values
  if (drift > 27) {                               //wrap protection for drift
    drift = 27;
  }
  if (drift < -27) {
    drift = -27;
  }
  Serial.print("    drift     ");
  Serial.println(drift);
  Forward(drift);                                 //drive forward
}

void Forward (int drift) {
  if (drift > 90) drift = 90;                     //wrap protection
  if (drift < -90) drift = -90;
  leftmotor.write(90 + drift);                    //drive left motor using
calculated drift value
  rightmotor.write(90 - drift);                   //drive right motor using
calculated drift value
}

void setIntake (int intakeLeftValue, int intakeRightValue) {    //function to
control intake speeds
  intakeMotorLeft.write(intakeLeftValue);
  intakeMotorRight.write(intakeRightValue);
}

void ArmRaise() {
  int error = constrain(analogRead(potPort) - potTarget, -90, 90);
//calculate difference between potentiometer value and desired height
  armMotorLeft.write(90 - Kp * error);                                  //use
proportional control to keep arm at fixed height
  armMotorRight.write(90 + Kp * error);
}
```

Autonomous Code 2: Drive on Ramp

```
#include<Encoder.h>
#include<Servo.h>
#include <LiquidCrystal.h>


int Kp = 1;
int potPort = A10;
int potTarget = 200;
int driveTarget = 800;
int Kp2 = 2;
int DK = 3;

Servo rightmotor;
Servo leftmotor;
Servo armMotorLeft;
Servo armMotorRight;

Servo intakeMotorLeft;
Servo intakeMotorRight;

Encoder encoder1(28, 29);
Encoder encoder2(24, 25);

const int leftMotorPin = 5;
const int rightMotorPin = 4;


LiquidCrystal lcd(40, 41, 42, 43, 44, 45);

void setup() {
  armMotorLeft.attach(7, 1000, 2000);
  armMotorRight.attach(6, 1000, 2000);
  leftmotor.attach(leftMotorPin, 1000, 2000);
  rightmotor.attach(rightMotorPin, 1000, 2000);
  intakeMotorLeft.attach(9, 1000, 2000);
  intakeMotorRight.attach(8, 1000, 2000);
  lcd.begin(16, 2);
}

void loop() {
  intakeMotorLeft.write(0);                                          //set
intake motors to constantly be spinning inwards to hold onto preloads
  intakeMotorRight.write(180);
  int error = constrain(analogRead(potPort) - potTarget, -90, 90);      //find
difference between potentiometer value and height of arm
```

```
  armMotorLeft.write(90 - Kp * error);                                    //use
proportional control to keep arm at fixed height
  armMotorRight.write(90 + Kp * error);
  int leftValue = encoder1.read();                                        //get
value of encoders
  int rightValue = encoder2.read();
  float distance = (((leftValue + rightValue) * -1 / 2));                 //use
average of the encoder values to calculate distance
  while (distance < 800 ) {                                               //while
robot is not yet at ramp, drive straight
    leftValue = encoder1.read();
    rightValue = encoder2.read();
    distance = (((leftValue + rightValue) * -1 / 2));                     //check
distance within while loop
    lcd.print("distance:  ");
    lcd.print(distance);
    int error2 = driveTarget - distance;
//calculate difference between input target and actual distance
    int output = Kp2 * error2;                                            //use
proportional control to control distance of robot
    int difference = abs(leftValue) - abs(rightValue);
//calculate difference between wheel speeds for drift correction
    if (difference > 20) {                                                //wrap
protection
      difference = 20;
    }
    if (difference < -20) {
      difference = -20;
    }
    lcd.print("    difference     ");
    lcd.print(difference);
    int drift = DK * difference;                                          //use
proportional control to control drifting of robot
    if (drift > 27) {                                                     //wrap
protection
      drift = 27;
    }
    if (drift < -27) {
      drift = -27;
    }
    Drive(output - drift);                                                //drive
motors at calculated seed
  }
  leftmotor.write(90);                                                    //stop
motors
```

```
  rightmotor.write(90);
  while (1)  {
//infinite loop to ensure motors stay stopped

  }
}

void Drive(int output) {
  if (output > 90) output = 90;                                    //wrap
protection
  if (output < -90) output = -90;
  leftmotor.write(90 + output);                                    //drive
motors at calculated speed
  rightmotor.write(90 - output);
}
```

Autonomous Code 3: Teleoperation

```
#include <DFW.h> // DFW include
#include <Servo.h> // servo library

int ledpindebug = 13; //Wireless controller Debug pin.
int potPort = A10;
int potTarget = 360;
DFW dfw(ledpindebug);
Servo rightmotor; // Servo object
Servo leftmotor; // Servo object
Servo armMotorLeft;
Servo armMotorRight;
Servo intakeMotorLeft;
Servo intakeMotorRight;


void setup() {
  dfw.begin(9600, 1);
  leftmotor.attach(5, 1000, 2000);
  rightmotor.attach(4, 1000, 2000);
  armMotorLeft.attach(7, 1000, 2000);
  armMotorRight.attach(6, 1000, 2000);
  intakeMotorLeft.attach(9, 1000, 2000);
  intakeMotorRight.attach(8, 1000, 2000);
  dfw.update();
  while (dfw.start() == 0) { //waits for controller to init
    Serial.println("init");
    dfw.update();
    delay(20);
  }

}

void loop() {
  dfw.update();// Called to update the controllers output. Do not call faster
than every 15ms.
  leftmotor.write(dfw.joysticklv());
  rightmotor.write(180 - dfw.joystickrv());
  if  (dfw.l2() == 1){                        //keep intake spinning forwards
while button not pressed
  intakeMotorRight.write(180);
  intakeMotorLeft.write(0);
  }
  if (dfw.r2() == 0) {                        //raise arm with right trigger
    armMotorRight.write(180);
    armMotorLeft.write(0);
```

```
    dfw.update();
  }
  if (dfw.r2() == 1) {                        //keep arm from moving if button not
pressed
    armMotorRight.write(90);
    armMotorLeft.write(90);
    dfw.update();
  }
  if (dfw.r1() == 0) {                        //lower arm with left trigger
    armMotorRight.write(0);
    armMotorLeft.write(180);
    dfw.update();
  }

  if (dfw.r1() == 1) {                        //keep arm from moving if button not
pressed
    armMotorRight.write(90);
    armMotorLeft.write(90);
    dfw.update();
  }

  if (dfw.l2() == 0) {                       //reverse direction of intake to score
eggs
    intakeMotorRight.write(0);
    intakeMotorLeft.write(180);
    dfw.update();
  }

  if(dfw.one() == 0) {                       //raise arm to specific height for
easier scoring in NEST
    int potValue = analogRead(potPort);
    while (potValue < potTarget){     //225
      armMotorRight.write(0);
      armMotorLeft.write(180);
      potValue = analogRead(potPort);
      if (potValue >= potTarget){
        armMotorRight.write(90);
        armMotorLeft.write(90);
        continue;
      }
    }
  }
}
```

Autonomous Code 4: Final RBE 1001 Template

```
/* This is the RBE 1001 Template as of 7/1/15 This Template
is designed to run the autonomous and teleop sections of the final
competition. Write and test your autonomous and teleop code on your
own, then simply copy paste your code into the appropriate section
and make sure the time passed to each function corresponds with the
time in seconds that each set of code should run. IE
autonomous(20); will run for 20 seconds after the transmitter is
turned on.The code will not start until the controller is turned on, connected,
and start is pressed.
There are print statements commented out that were used to test */

#include <DFW.h> // DFW include
#include <Servo.h> // servo library
#include <Encoder.h>

int ledpindebug = 13; //Wireless controller Debug pin. If lit then there is no
communication.
int potPort = A10;
int potTarget = 380;
int potTarget2 = 200;
//DFW dfw(ledpindebug);  // Instantiates the DFW object and setting the debug
pin. The debug pin will be set high if no communication is seen after 2 seconds
Servo rightmotor;
Servo leftmotor;
Servo armMotorLeft;
Servo armMotorRight;
Servo intakeMotorLeft;
Servo intakeMotorRight;

DFW dfw(ledpindebug);  // Instantiates the DFW object and setting the debug pin.
The debug pin will be set high if no communication is seen after 2 seconds


int lineFollowerLeftPort = A0;
int lineFollowerRightPort = A1;
int lineFollowerLeft;
int lineFollowerRight;
int limitSwitchPort = 22;
int limitSwitchPort2 = 23;
int limitSwitch2;
int limitSwitch;
int Kp = 1;


Encoder encoder1(28, 29);
```

```
Encoder encoder2(24, 25);



void setup() {
  Serial.begin(9600); // Serial output begin. Only needed for debug
  dfw.begin(9600, 1); // Serial1 output begin for DFW library. Buad and port #.
"Serial1 only"
  leftmotor.attach(4, 1000, 2000); // left drive motor pin#, pulse time for 0,
pulse time for 180
  rightmotor.attach(5, 1000, 2000); // right drive motor pin#, pulse time for 0,
pulse time for 180
  //Serial.println("start");

  while (dfw.start() == 0) { //waits for controller to init
    Serial.println("init");
    dfw.update();
    delay(20);
  }

  leftmotor.attach(5, 1000, 2000); // left drive motor pin#, pulse time for 0,
pulse time for 180
  rightmotor.attach(4, 1000, 2000); // right drive motor pin#, pulse time for 0,
pulse time for 180
  armMotorLeft.attach(7, 1000, 2000);
  armMotorRight.attach(6, 1000, 2000);
  intakeMotorLeft.attach(9, 1000, 2000);
  intakeMotorRight.attach(8, 1000, 2000);
  // put your setup code here, to run once:

}

void autonomous(volatile unsigned long time) // function definition
{

  while (dfw.start() == 1) { // waits for start button
    Serial.println("waiting for start");
    dfw.update();
    delay(20);
  }

  unsigned long startTime = millis(); // sets start time of autonomous
  time = time * 1000; // modifies milliseconds to seconds
  while ((millis() - startTime <= time) && (dfw.select())) // compares start time
to time entered in the autonomous function and
```

```
  {
    // The select button can be used to skip the autonomous code.
    // Enter Autonomous User Code Here
    intakeMotorLeft.write(0);
  intakeMotorRight.write(180);
  int error = constrain(analogRead(potPort) - potTarget2, -90, 90);
  armMotorLeft.write(90 - Kp * error);
  armMotorRight.write(90 + Kp * error);
  int leftValue = encoder1.read();
  int rightValue = encoder2.read();
  float distance = (((leftValue + rightValue) * -1 / 2));
  if (distance < 900) {
    drive(135, 38);
    int leftValue = encoder1.read();
    int rightValue = encoder2.read();
  }
  else {
    drive(90, 90);
  }
    Serial.println("Autonomous"); //prints Autonomous over serial (usb com port)

    dfw.update();//used for autonoumous skip
    delay(20); //delay to prevent spamming the serial port and to keep servo and
dfw libraries happy

  }
}

void drive (int leftValue, int rightValue) {
  leftmotor.write(leftValue);
  rightmotor.write(rightValue);
}

void teleop(unsigned long time) { // function definition
  unsigned long startTime2 = millis(); // sets start time of teleop
  time = time * 1000; // modifies milliseconds to seconds
  while (millis() - startTime2 <= time) // compares start time to time entered in
the teleop function
  {
    //tank drive code next 4 lines
    dfw.update();// Called to update the controllers output. Do not call faster
than every 15ms.
    //rightmotor.write(dfw.joystickrv());     //DFW.joystick will return 0-180 as
an int into rightmotor.write
    //leftmotor.write(180 - dfw.joysticklv());    //DFW.joystick will return
```

```
0-180 as an int into leftmotor.write
    //delay(20); // Servos do not like to be called faster than every 20 ms. Also
the Serial data is sent every 15ms

    // Enter Teleop User Code Here
  leftmotor.write(dfw.joysticklv());   //DFW.joystick will return 0-180 as an int
into leftmotor.write
  rightmotor.write(180 - dfw.joystickrv());      //DFW.joystick will return 0-180
as an int into rightmotor.write
  if  (dfw.l2() == 1){                          //keep intake spinning forwards
while button not pressed
  intakeMotorRight.write(180);
  intakeMotorLeft.write(0);
  }
  if (dfw.r2() == 0) {                        //raise arm with right trigger
    armMotorRight.write(180);
    armMotorLeft.write(0);
    dfw.update();
  }
  if (dfw.r2() == 1) {                        //keep arm from moving if button not
pressed
    armMotorRight.write(90);
    armMotorLeft.write(90);
    dfw.update();
  }
  if (dfw.r1() == 0) {                        //lower arm with left trigger
    armMotorRight.write(0);
    armMotorLeft.write(180);
    dfw.update();
  }

  if (dfw.r1() == 1) {                       //keep arm from moving if button not
pressed
    armMotorRight.write(90);
    armMotorLeft.write(90);
    dfw.update();
  }

  if (dfw.l2() == 0) {                       //reverse direction of intake to
score eggs
    intakeMotorRight.write(0);
    intakeMotorLeft.write(180);
    dfw.update();
  }
```

```
  if(dfw.one() == 0) {                    //raise arm to specific height for
easier scoring in NEST
    int potValue = analogRead(potPort);
    while (potValue < potTarget){     //225
      armMotorRight.write(0);
      armMotorLeft.write(180);
      potValue = analogRead(potPort);
      if (potValue >= potTarget){
        armMotorRight.write(90);
        armMotorLeft.write(90);
        continue;
      }
    }
  }

    Serial.println("TeleOp"); //prints Teleop over serial (usb com port)
    delay(20); //delay to prevent spamming the serial port

  }
  exit(0); // exits program
}


void loop() {

  autonomous(20); //time in seconds to run autonomous code

  teleop(180); //time in seconds that teleop code will run
}
```

Motor Data Charts:

VEX 3-Wire Motors:

Figure A.1

Speed (rpm)	Torque (N m)	Torque (in lbs	Current (A)	Power (wt)	Efficiency	Heat (wt)
0	0.49	4.31	1.900	0.0	0%	15
7	0.45	4.02	1.784	0.4	3%	14
15	0.42	3.74	1.668	0.7	5%	12
22	0.39	3.45	1.552	0.9	8%	11
30	0.36	3.16	1.436	1.1	10%	10
37	0.32	2.87	1.320	1.3	12%	9
45	0.29	2.59	1.204	1.4	15%	8
52	0.26	2.30	1.088	1.4	17%	7
60	0.23	2.01	0.972	1.4	19%	6
67	0.19	1.72	0.856	1.4	21%	5
75	0.16	1.44	0.740	1.3	22%	5
82	0.13	1.15	0.624	1.1	23%	4
90	0.10	0.86	0.508	0.9	23%	3
97	0.06	0.57	0.392	0.7	22%	2
105	0.03	0.29	0.276	0.4	17%	2
112	0.00	0.00	0.160	0.0	0%	1

Desired Volt	7.8 V
Ref Volt	7.8 V
Ref Free Spd	112 RPM
Ref Stall Torq	4.31 in-lbs
Ref Stall Cur	1.9 A
Ref Free Cur	0.16 A

Figure A.1: VEX 3-Wire Motor Data

VEX 393 Motors:

Figure A.2

Speed (rpm)	Torque (N m)	Torque (in lbs)	Current (A)	Power (wt)	Efficiency	Heat (wt)
0	1.67	14.76	4.800	0.0	0%	35
7	1.56	13.78	4.505	1.1	3%	31
13	1.45	12.79	4.209	2.0	7%	28
20	1.33	11.81	3.914	2.8	10%	25
27	1.22	10.82	3.619	3.4	13%	23
33	1.11	9.84	3.323	3.9	16%	20
40	1.00	8.86	3.028	4.2	19%	18
47	0.89	7.87	2.733	4.3	22%	15
53	0.78	6.89	2.437	4.3	25%	13
60	0.67	5.90	2.142	4.2	27%	11
67	0.56	4.92	1.847	3.9	29%	9
73	0.44	3.94	1.551	3.4	31%	8
80	0.33	2.95	1.256	2.8	31%	6
87	0.22	1.97	0.961	2.0	29%	5
93	0.11	0.98	0.665	1.1	23%	4
100	0.00	0.00	0.370	0.0	0%	3

Desired Volt	7.2 V
Ref Volt	7.2 V
Ref Free Spd	100 RPM
Ref Stall Torq	14.76 in-lbs
Ref Stall Cur	4.8 A
Ref Free Cur	0.37 A

Figure A.2: VEX 393 Motor Data

Scoring Guidelines:

Table 3

Scoring Guidelines	
Autonomous Scores	**End of match**
Robot supported by RAMP: 5pts	Egg in COOP: 1 pt each
PEN outside vertical projection: 5pts	Egg in PEN: 3 pts each
Each of the first two eggs in COOP: 1pt each	Egg in NEST: 6 pts each
Each of the first two eggs in PEN: 3pts each	Robot not touching carpet: 5pts
Each of the first two eggs in NEST: 6pts each	Robot supported only by a PERCH: 30pts

Table 3: Scoring guidelines for project

Pictures:

Figure A.3

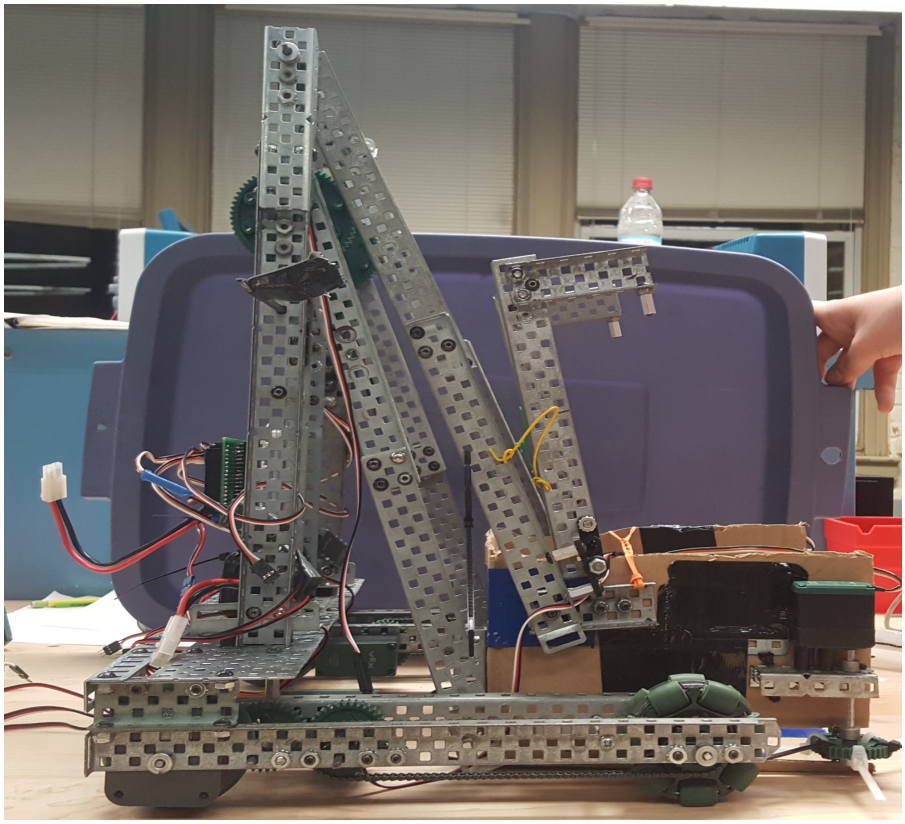


Table A.3: Final robot side

Figure A.4

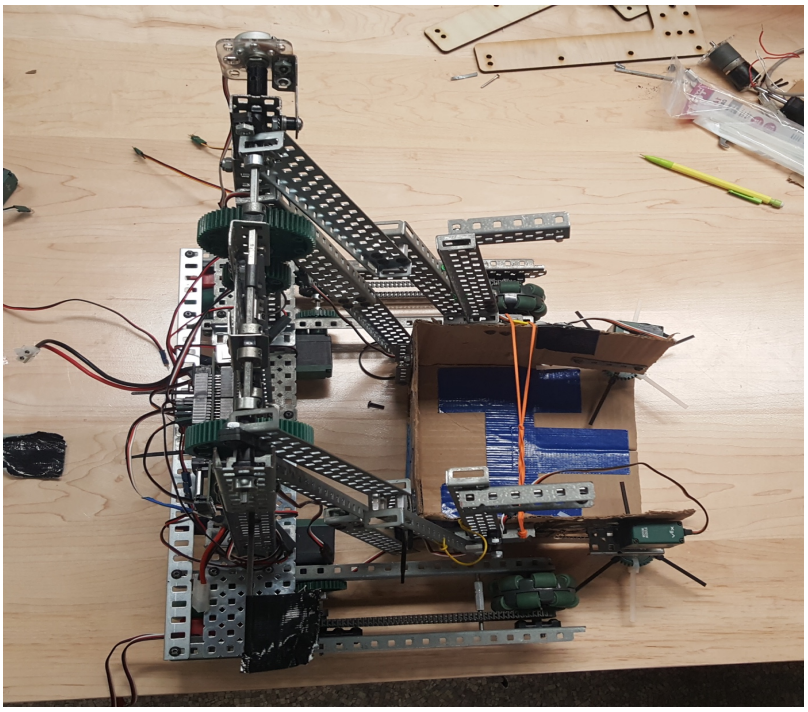


Table A.4: Final robot above

Figure A.5

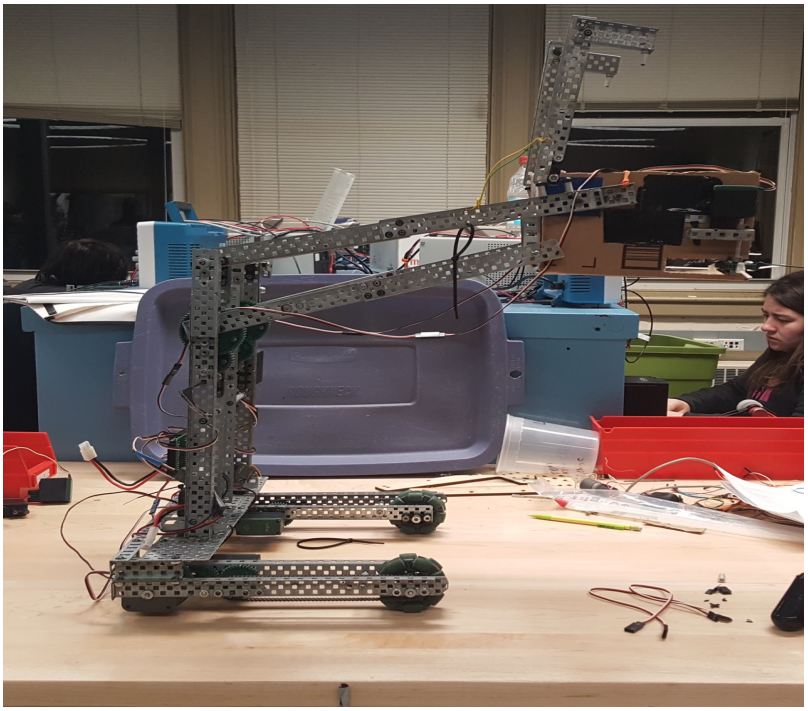


Table A.5: Final robot raised